MSD SUPER DISK DRIVE OPERATING MANUAL

TABLE OF CONTENTS

INTRODUCTION

The **MSD SUPER DISK DRIVE** is perhaps the most versatile and efficient disk drive built for the Commodore series of personal and business computers. The MSD SUPER DISK DRIVE is unique in that it features both serial and IEEE bus interfaces in one unit. The result of this flexibility is that the **MSD SUPER DISK DRIVE** is compatible with the Commodore SERIES 2001 (with BASIC Version 3.0 or higher), SERIES 3000 (with BASIC Version 3.0 or higher), SERIES 4000 (with BASIC Version 4.0), SERIES 8000 (with BASIC Version 4.0), COMMODORE 64 and VIC 20 computers. The wide range of compatibility is accomplished by providing a dual serial computer interface and an IEEE parallel interface. These two interfacing methods allow multiple disk drives and printers to share the bus concurrently. The use of the parallel interface provides for significantly faster data transfers over the serial bus interface. With the addition of MSD's optional CIE or VIE interface board (IEEE interfaces for Commodore 64 or VIC 20) that plugs into the expansion port of the Commodore 64 or VIC 20 respectively, the user is even able to obtain this parallel speed enhancment on the Commodore 64 or VIC 20. The versatility is enhanced even more by the fact that the **MSD SUPER DISK DRIVE** is upward compatible with the Commodore 2040 disk drive and read/write compatible with the Commodore 2031, 4040, 1540 and 1541 disk drives. This means that disks that were saved from the 2040 can be read by the **MSD SUPER DISK DRIVE** and disks that have been created on the 2031, 4040, 1540 and 1541 can be both read from and written to without any harm to existing programs on the disk.

The **MSD SUPER DISK DRIVE** is a high quality 'smart' disk drive that does not require the use of any memory in the computer. Instead, the **SUPER DISK DRIVE** contains its own microprocessor and memory allowing the computer to transfer commands to the disk and then continue other operations. The disk in turn processes those commands to perform the desired function. In order to be compatible with disks created on the various Commodore disk drives, the **SUPER DISK DRIVE** uses 35 tracks on a disk with a density of 48 TPI (tracks per inch). The single-head drive provides a capacity of 174,848 bytes on a single diskette. These bytes are arranged in blocks with each block representing one sector or 256 bytes. Figure I.1 provides more details on the technical specifications pertaining to the **MSD SUPER DISK DRIVE.**

This manual is designed to give the user the necessary information for normal usage of the disk drive as well as advanced information for the programmer that wants the utmost in disk drive capability. This manual begins with the actual connection of the **MSD SUPER DISK DRIVE** in Chapter 1. Chapter 2 deals with the normal operations such as saving and loading programs. The more advanced applications are covered in Chapter 3 while Chapter 4 deals with the programming of the disk controller and its memory. Chapter 5 explains the methods for changing the disk device number. An appendix section is provided to give a list of error messages as

well as a summary of all commands available to the user including the BASIC 4.Ø commands. It is very important that at least Chapters 1 and 2 of this manual be read before attempting to use the disk drive in order to avoid any problems that may be encountered during the use of your **MSD SUPER DISK DRIVE.**

FIGURE I.1. MSD SUPER DISK DRIVE TECHNICAL SPECIFICATIONS

****

STORAGE (NOTE 1)

Total Capacity	174848 bytes per diskette
Sequential	168656 bytes per diskette
Relative	167132 bytes per diskette
	65535 records per file
Directory Entries	144 per diskette
Blocks	683 total per diskette
	664 available per diskette
Tracks	35 per diskette
Sectors	17 to 21 per track
Bytes	256 per sector
Diskettes	Standard 5-1/4", single sided, single density

NOTE 1: The SD-2 contains two disk drive mechanisms and can therefore handle two times the above capacities (One for each diskette).

SOFTWARE

16K Bytes Operating System
4K RAM buffer area (6K for the SD-2)
Microprocessor based disk controller (6511Q)
Commodore Compatible Serial Bus Interface
Commodore Compatible IEEE Parallel Bus Interface

INTERFACE

Dual Commodore compatible Serial Bus
Commodore compatible IEEE Parallel Bus
Jumpers for selecting the device number

PHYSICAL

Dimensions	SD-1		SD-2	
Height	6.2"	(157 mm)	6.2"	(157 mm)
Width	4.2"	(107 mm)	5.9"	(150 mm)
Depth	13.3"	(338 mm)	13.3"	(338 mm)

Electrical requirements
Voltage 110 or optional 220 VAC
Frequency 50 or 60 Hertz
Power 50 Watts

****

CHAPTER 1
INSTALLATION INSTRUCTIONS

The **MSD SUPER DISK DRIVE** consists of a disk drive, power cable, serial bus cable and this manual. Figure 1.1 should be used in the following steps to help locate the various positions of components.

1) Connect the power cable between the disk drive and a 110 VAC electrical outlet. One end of the power cable has a connector that plugs into the rear of the **SUPER DISK DRIVE.** The other end of the power cable comes with a three-pronged grounded electrical plug that should be plugged into the the wall. **After plugging the power cord into the disk drive and the wall, check to make sure that no sounds come from the disk and the lights on the front of the drive are OFF.** If they are not, switch the power switch on the back of the disk drive to off. Make sure that the power is OFF on all the peripheral devices to be connected to the computer.

2) Connect the disk drive to the Commodore computer. The **MSD SUPER DISK DRIVE** has two possible methods of being connected to the computer. When using a VIC 20 or Commodore 64, the disk drive can be connected by means of either the serial bus cable or if a CIE or VIE cartridge is being used, by means of an optional PET - IEEE bus cable. All other Commodore computers should be connected using an optional PET - IEEE bus cable. Follow the appropriate instructions below for either serial or parallel connection.

a) Serial Bus Interface Connection - The **MSD SUPER DISK DRIVE** comes with a serial interface cable that has a 6 pin DIN type connector on each end. One end is plugged into the Commodore 64 or VIC 20 while the other end is plugged into either of the two serial connectors on the back of the disk drive. When connecting more than one disk drive and/or printer on the serial bus, use the other available DIN socket on the back of the disk drive to 'daisy chain' the devices together. The disk drives should be the first devices connected to the computer if more than one device is to be used in the 'daisy-chain'. **Make sure the power is off to all devices before connecting cables.**

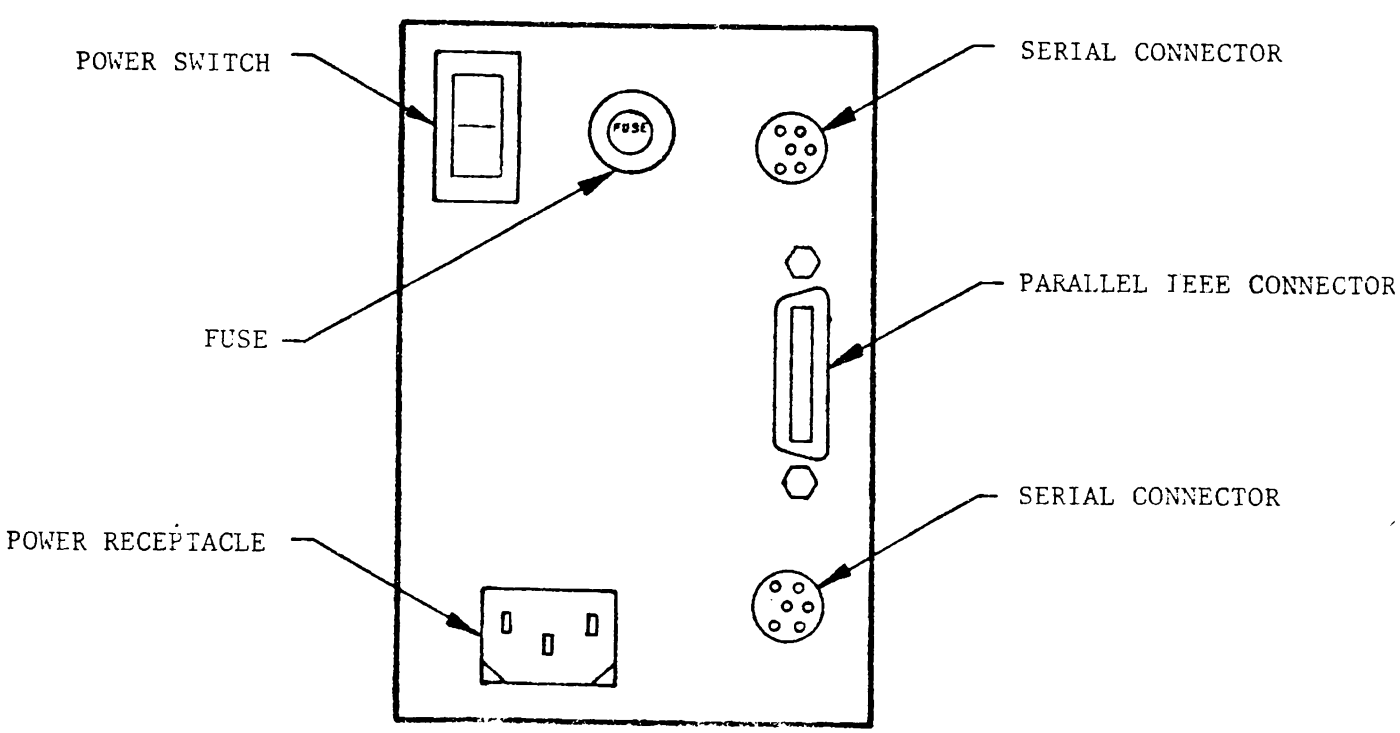
b) Parallel Bus Interface Connection - The MSD SUPER DISK DRIVE also features an IEEE parallel bus interface for computers that are equipped with that interface configuration. The following computers have the IEEE interface instead of the serial.

Series 2001 (with BASIC 3.0)
Series 3000 (with BASIC 3.0)
Series 4000 (with BASIC 4.0)
Series 8000 (with BASIC 4.0)

In addition the Commodore 64 and VIC 20 can be used with the IEEE parallel interface when equipped with the appropriate IEEE adapter cartridge such as the CIE and VIE manufactured by MSD. The use of the IEEE interface provides for a significantly faster data transfer rate over the serial interface. In order to use the IEEE interface an optional IEEE cable must be purchased. The cable connecting a disk drive to the computer is referred to as a PET - IEEE cable and is available from MSD or dealers selling Commodore peripheral products. If more than one drive or printer is to be connected ('daisy-chained') to the IEEE bus, then an IEEE - IEEE cable will be needed between each of the devices. Disk drives should be connected to the computer first when 'daisy-chaining' devices together. **Again make sure all power is OFF to the devices.** Connect the flat edge-connecter type cable end to the computer with the connector lettering facing up and the IEEE type connector to the rear of the disk drive. Additional IEEE devices can be connected by plugging 'piggyback' onto the back of the MSD drive cable and then to the other IEEE device.

When using more than one disk drive, it is necessary to change the device number of one of the drives. Chapter 5 covers the procedure for changing the drive device number through software or hardware.

FIGURE 1.1. CONNECTOR LOCATIONS ON THE MSD SUPER DISK DRIVE



CHAPTER 2
OPERATING INSTRUCTIONS

After the **MSD SUPER DISK DRIVE** has been properly connected to the computer and AC outlet, the system is ready for operation. Specific procedures should be followed each time the system is to be powered ON or OFF. The following procedures should be strictly followed:

a) Make sure no diskette is in the drive at the time that power is turned ON or OFF.
b) Follow the precautions explained in the box on care of diskettes.
c) Turn ON all peripheral devices before turning ON the computer. The computer contains circuitry that resets all peripherals on power-up making sure that they are ready for operation.
d) Make sure the top light (LED) on the disk drive glows green after the computer has been powered ON. If for some reason the light flashes for more than five seconds, the power should be turned OFF. Try the procedure again. If the top LED still flashes, the power-on self tests have found a failure within the MSD SUPER DISK DRIVE. The number of times that the LED flashes continuously between pauses will indicate what has failed in the drive. The following table indicates which faulty component is associated with the respective number of flashes:

Number of Flashes	Component
1	Microprocessor U7
2	RAM U1
3	RAM U2
4	RAM U3
5	ROM U5
6	ROM U6
7	Drive Mechanism

If the LED flashes with a continuous flash with no pauses, the disk drive may be able to be reset under software control to clear the error condition. The following instruction can be entered from the keyboard to reset the disk drive:

```
OPEN 15,8,15,"UJ":CLOSE 15
```

NOTE: The dual disk drive (SD-2) powers up in a DRIVE Ø/DRIVE 1 configuration as device number 8 unless the user has reconfigured the hardware of the disk drive as explained in Chapter 5. The SD-1 will only respond as DRIVE Ø but the device number can be changed from the factory configured device number 8 as explained in Chapter 5. The SD-2 will have the green LED on and both DRIVE Ø and DRIVE 1 LEDs will be off if the drive is ready for operation.

If the above command does not cause the LED to stop flashing, the

disk drive or, in the case where more than one peripheral is connected to the computer, some other peripheral is defective. Remove any other peripherals from the bus to determine where the failure has occurred. If the disk drive still fails, return it for repair or replacement.

If the disk drive LED never comes on, some failure in the power supply has occurred. After turning off all equipment and unplugging the disk drive from the AC outlet, check the fuse to make sure it is good. If the fuse continuously blows or no problem can be found, return the disk drive for repair or replacement.

After the LED on the disk drive is green, the drive is ready for use. The disk drive uses a rotating lever to lock the diskette inside the drive. When the lever is across the diskette opening, diskettes cannot be removed or inserted. Gently turning the lever 90 degrees (open position) will allow insertion or removal of a diskette. The disk drive will not operate with the lever in the open position.

CARE OF DISKETTES

Diskettes contain valuable data that can be easily destroyed if proper precautions are not used. The following steps will help to protect the diskettes and the data that is on those diskettes :

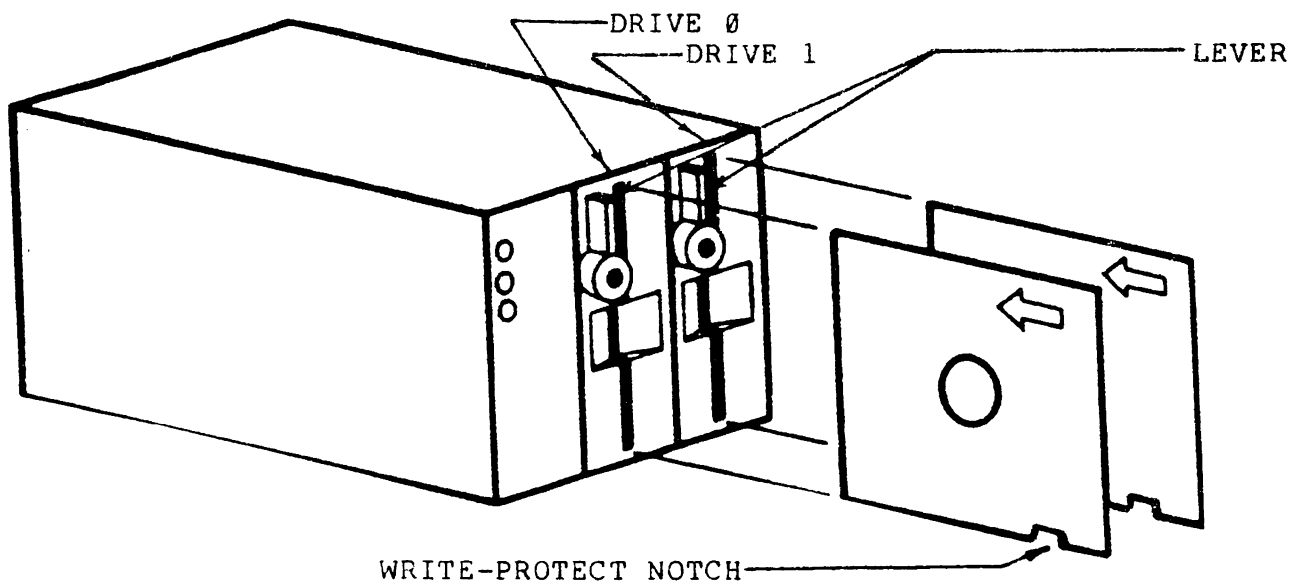
1) Keep the diskettes away from any source of magnetic fields such as magnets, motors, etc.
2) Keep the diskette in the protective jacket when it is not in the disk drive.
3) Do not touch the diskette surface that is present in the opening of the disk.
4) Do not bend or fold the diskette and do not write on the diskette or the diskette jacket if a disk is stored in the jacket.
5) Do not expose the diskette to heat or sunlight. Because computers and peripherals often get quite warm, they can harm diskettes if the diskette is left on the computer or peripheral.
6) Always insert the diskette into the disk drive carefully making sure the power is ON before inserting or removing a diskette.

Examine a diskette at this time to become familiar with the physical characteristics of the diskette. Especially read the Care of Diskettes box above. Figure 2.1 points out the important areas of the diskette.

The write-protect notch on the diskette should be pointed down

when inserting the diskette into the disk drive. After inserting the diskette into the drive, turn the lever on the front of the disk drive until it covers the diskette opening. The lever should turn freely. If it does not, do not force the lever. The lever has stops to prevent it from turning more than 90 degrees and the lever will not close correctly if the diskette is not inserted completely into the drive. Remove the diskette and reinsert it if the lever does not close easily.

FIGURE 2.1. DESCRIPTION OF DISKETTE



Before actually using the **MSD SUPER DISK DRIVE** it is helpful to understand some of the terminology used when working with computers and data. The next section deals with an explanation of some of the more common terms. This section is more for the newcomer to computers and disk drives but should be reviewed by the advanced programmer also since some terms have been used differently in various application.

DISK OPERATING SYSTEM - The disk operating system is the program within the **MSD SUPER DISK DRIVE** that allocates storage on the diskette as it is needed, locates the desired data on the diskette and maintains all of the operations concerned with computer-disk communications and disk-diskette communications.

DIRECTORY - The directory is a list of all programs and other files that have been stored on a particular diskette. The directory is maintained by the disk drive on the diskette itself and is updated every time a program is saved or a file has been opened for writing. Each diskette can have up to 144 entries in the directory. The directory listing also contains a number next to each file which indicates how many blocks of diskette are taken up by that program. The last line of the directory listing

indicates how many blocks are still available on the diskette for additional programs or data files.

BAM - The BAM (block availibility map) is a place physically located in the center of the diskette that is used by the disk operating system to determine which blocks of the diskette have been used and which are available for additional storage. Each time a program is saved or file is closed to the diskette, the operating system checks the BAM to see what is available. After the program is saved or the file is closed, the BAM is updated to reflect the diskette space that has been used. If the diskette does not have enough space for the file the LED on the front panel will flash and an error message will be generated.

BLOCKS - Each diskette is divided into 683 blocks at the time the disk is formatted. Blocks have no relationship to programs or data files other than the fact that a program or data file will always be at least one block in length. Instead, a block is used as a measure of the capacity of the disk. Each block represents 256 bytes of data (this is the same size as a sector). Of the 683 blocks of storage on each diskette, 19 are used by the system for things like the directory and BAM. The remaining 664 blocks are available for the storing of programs and data by the user.

TRACKS - Tracks are physical divisons on the diskette that contain data. A track is a ring around the diskette that contains a fixed number of sectors within that track. Each groove on a phonograph record would be similar to a track on the diskette. As the diskette head moves along the surface of the diskette it stops at a track specified by the disk operating system based on the location of a particular file on the diskette. The track closest to the outside of the diskette is the first track and also the largest track. The tracks decrease in size as they move towards the center of the disk. Each diskette is divided into 35 tracks.

SECTORS - Sectors are the smallest divisible section of the diskette for storage. Each sector (or block) is 256 bytes in length. Each track will contain from 17 to 21 sectors depending on whether the track is close to the outside or the inside of the diskette. Since the outside tracks have a larger circumference, more sectors can be stored on those tracks. Only 254 bytes of each sector can be used for actual data storage. The other two bytes are used to point to the next track and sector of the file or to indicate the end of the file.

FILES - Where tracks and sectors represent the physical storage of data on the diskette, files represent the logical storage of data on the diskette. The disk drive itself does not care what data is stored on the disk but the data must be stored in a manner that will allow the user to transfer the data from the disk to the computer in the same way as it was first stored to the disk. For this reason, the disk operating system maintains the data in the form of files.

WRITE-PROTECT - Each diskette has the ability to be protected so that data cannot be inadvertantly written to the diskette. This protection is accomplished by covering a notch on the diskette known as the write-protect notch (See Figure 2.1). If this notch is covered and an attempt is made to write data to the diskette, an error will be generated.

The **MSD SUPER DISK DRIVE** is now ready for operation. Because the MSD drive is a smart peripheral, it recognizes a large number of commands. These commands have been broken into several different sections to make for easier understanding and reference. The first section deals with the most frequently used commands while the second section explains the utility commands that are available for the user. The last section allows the user to take advantage of the advanced commands that are available within the **MSD SUPER DISK DRIVE.** Be sure and read Section 2.1 since a number of special features and rules are explained as well as the commands themselves.

SECTION 2.1 - Frequently Used Commands

The most frequently used commands are those commands that will load and save files on a disk that has already been formatted. Prepackaged programs fall into this category but any new disk that has never had anything stored on it will have to be formatted before it can be used. Refer to Section 2.2 of this chapter for the appropriate utility commands for initializing and formatting a diskette.

LOAD COMMAND: **LOAD <program name>,<device #>,<secondary address>**

Purpose: Getting a program from the diskette into the computer memory

The disk drive responds to certain commands that must be carefully entered in order for the commands to operate correctly. In this case, the command is LOAD which will transfer a program from diskette to the computer memory. The program on diskette is not altered. The program name can be a string variable or a filename that has been enclosed in quotes. The string variable must have been defined in an earlier statement. **The < and > symbols are not a part of the command. They serve to indicate that the programmer needs to fill in the portion of the command that is inside those symbols.** The following example is a LOAD with a string variable for the program name:

```
LOAD A$,B
```

where A$ has been previously defined as a file that is in the directory. Notice that B must be a variable that has been defined as 8 if the device number has not been changed. The example below illustrates the more common method of specifying a filename inside of quotes:

LOAD "EXAMPLE",8

Notice the command above is LOAD which tells the specified device to bring a file into the computer. The actual file name to be loaded into the computer is specified inside the quotation marks. This file name will be the same as the name found in the directory (see next paragraph for the method of obtaining a directory) unless some special pattern matching techniques are used as will be explained later. The 8 specifies that the file is to be loaded from the **MSD SUPER DISK DRIVE.** All commands are terminated by pressing the RETURN key in order for the computer to know the command is ready to be processed.

The commands are directed to the disk drive by specifying the device number in the command. The device number like the program name can be a variable or a fixed value. The **MSD SUPER DISK DRIVE** comes configured for the device number to be 8 although this can be changed in software or hardware (See Chapter 5). The secondary address is optional and can be a variable or a fixed value. Specifying a Ø or not specifying a secondary address at all will load the program to the start of the BASIC memory area of the computer. Specifying a 1 for the secondary address will load the program to the same location that it was saved from (See the instructions for the SAVE command). Unless specially instructed, prepackaged software does not need a secondary address.

In order to examine the directory of a iskette, the directory must be loaded just as any program would be. The syntax for loading a directory into the computer is:

LOAD "$Ø",8

Notice that the directory is designated as $Ø inside the quote. Actually the Ø is only a designator to specify which disk drive in a dual disk drive configuration is being used when both drives have the same device number. The SD-1 is a single drive unit, and it is not necessary to specify the drive number in this case. It is a good practice however to always precede a file name with the Ø: since many operations must have the drive number specified in order to produce the desired results. The SD-2 is a dual drive unit and as such, responds to both a Ø and a 1 for the drive number. If no number is specified after the $, both directories will be loaded into the computer. If only the directory of a particular drive number is desired, then put the desired drive number after the $. Loading of the directory will erase any program that is currently in the computers memory.

After entering a LOAD command, the disk drive will turn on and try to find the file on the disk. The computer will display:

SEARCHING FOR specified program name

If the program is not found an error message will be generated and the LED will flash on the disk drive. If the file is found, the

computer will display:

FOUND specified program name

LOADING

If the file successfully loads into the computer, the computer will respond with:

READY

Any time the computer is finished with an operation, it will respond with READY and a flashing cursor. This indicates that additional commands can now be executed such as LIST or RUN. If the directory has been loaded, it can be displayed by using the LIST command followed by a RETURN key. Programs can be executed by entering the RUN command followed by a RETURN key.

It is possible to use some pattern matching techniques or wild cards when loading files from the disk. Pattern matching is a technique of using a single special character to replace some number of characters in a file name. The special character is an *. Below are some examples of pattern matching and their appropriate explanation:

LOAD "Ø:*",8

The above syntax will load the first file on the diskette. If the Ø: is left off, the last program accessed by the drive will be loaded unless no file has ever been accessed. In that case, the first file of the program will be loaded. This command must be carefully used. If a file has been previously loaded from a diskette and a new diskette is inserted that does not have that previous file on it, using the * by itself inside the quotes in the LOAD command will require the disk to be reset since it will not find that previous file.

LOAD "Ø:TE*",8

The above example will load the first file in the directory that begins with TE.

Pattern matching can also be used in searching the directory. For example, the following command will find all files on drive Ø that begin with a T:

LOAD "$Ø:T*",8

By entering a LIST followed by a RETURN, only those files on the diskette that begin with a T will be listed.

Another method of loading files into the computer involves the use of a wild card. Just as some card games allow a card to be used to represent any other card, the question mark (?) can be used as a

wild card in the file name. The following examples indicate some of the uses of a wild card:

```
LOAD "Ø:TE??",8
```

In the above case, the disk drive will look for the first file on drive Ø of the disk that has four letters and begins with TE. Notice that the disk will not care what the last two letters of the file are. If three files in the directory are called TEX, TEST, and TEXT in that order, the above command will load the program TEST (the program must be four letters long). Any letter can be replaced by a wild card.

The wild card can also be used for displaying certain directory entries. For example :

```
LOAD "$Ø:TE?T",8
```

will find any files in the directory of drive Ø that are four letters long and contain TE for the first two letters and T for the last letter.

SAVE COMMAND: **SAVE <program name>,<device #>**

Purpose: To put a copy of the program in the computer memory onto the diskette that is in the disk drive.

The syntax for the SAVE command is very similar to the LOAD command. The program name can be a string variable or an actual file name inside of quotation marks. The device number may be a variable or a fixed value and should be 8 unless the device number has been changed. The secondary address is not used. The program will be saved from the start of the BASIC memory area to the end of the program unless a file is already on the disk that has the same file name or there is not sufficient space on the disk. If either of the last two conditions occur, the LED will flash indicating an error has occurred and an error message will be generated (See Appendix B for an explanation of error messages).

It is possible to replace a file on the diskette even if a file by that name exists on the diskette. This is common when a program is loaded, modified and then must be resaved over the old file. One possible way is to erase (SCRATCH) the old file and then SAVE the new file, but that is not very convenient. Instead of erasing the file, the @ character can be used to tell the disk drive to replace the existing file on the diskette with the program in the computer memory using the same file name. The syntax for replacing a file on disk with the program in the computer memory is:

```
SAVE "@Ø:program name",8
```

An example of the command would be:

SAVE "@Ø:TEST",8

This command will find the program TEST on drive Ø of the diskette if it exists and deletes it from the directory and BAM. A new entry is then created with the same file name of TEST and the program in memory is saved under that file name.

VERIFY COMMAND: **VERIFY <program name>,<device number>**

Purpose: The VERIFY command can compare an existing program on diskette with the program in computer memory to make sure they are identical.

This command is very useful for comparing a program in memory with one that is on disk to see if they are the same or for making sure that a program in memory has been stored to disk in its current form. The VERIFY command operates just like the LOAD command except the program is not loaded into the memory. Instead, it is compared to the contents of memory. The command must have the program name in the form of a variable string or an actual file name inside of quotes. The device number can either be a predefined variable or a fixed value (generally 8).

SECTION 2.2 - Utility Commands

The **MSD SUPER DISK DRIVE** contains a number of commands that are used to perform special tasks within the disk drive. Some of these tasks would be formatting a new diskette so that programs can be stored on them, erasing a program from the diskette or copying a program from one file name to another. These types of operations are called utilities. In order to use the utility commands, it is necessary to first establish a channel of communications between the disk drive and the computer. This channel of communications allows the computer and the disk drive to exchange information. The syntax for opening a channel is:

OPEN <file #>,<device #>,<secondary address>,<text string>

The file number can be any number between 1 and 255 but numbers greater than 127 may cause a linefeed to be sent after a return character. This number is assigned by the user and is used throughout communications between the computer and the disk to identify which file is being accessed.

The device number is 8 unless it has been changed in software or hardware.

The secondary address can be any number between Ø and 15, but some of the numbers are predefined. Secondary addresses Ø and 1 are reserved for the operating system during loading and saving. Secondary address 15 is reserved for the command channel as will

be explained later. The remaining secondary addresses are available for the user to transfer data to or from files. The same secondary address cannot be used when multiple files are opened and it is not possible to open the same file number with different secondary addresses.

The text string is a string that will cause a file on the disk by that name to be created, if it does not exist, and opened to allow transfer of data to or from that file on the diskette. A file on the disk cannot be opened unless the file name is specified in the text string. For the utility commands, the text string can be the actual command or it can be omitted completely.

Once a file has been OPENed, it will remain OPEN until a CLOSE command is executed for that channel, the drive is initialized, or an error condition causes the system to think the file is closed. If a file is already OPENed and an attempt is made to OPEN it again, a 'FILE OPEN ERROR' will be displayed.

Variables can be used for any of the portions of the OPEN command as long as they have been predefined. Some examples of OPEN statements are given below:

```
OPEN 5,8,5,"TEXT"

OPEN 15,8,15,"I0"

OPEN A,B,C,A$
```

Notice that the second example above is an example of a command being sent to the disk drive because the command channel is being used. The command is placed inside of quotes as a text string.

It is possible to open the command channel without sending the text string message. The text string can be sent to the disk with the use of a PRINT# command. The syntax for the PRINT# command is:

PRINT#<file #>,<text string>

The file number must be opened before using this command or an error will result. The text string can be a variable or text inside of quotes. In the case of the command channel, it would be a command. In the case of a channel that has been opened for data transfer, it will send that text string to the file that has been opened and it will be stored on the diskette. The explanation of transferring data using the PRINT# command will be described in more detail later.

The command channel (file number 15) has its own set of commands for accomplishing different utilities. These commands are transfered as mentioned above through the OPEN command or by the PRINT# command after the command channel has been opened. The syntax for sending the utility commands to the disk is:

OPEN 15,8,15,<command>

unless the command channel has already been opened in which case the syntax would be:

PRINT#15,<command>

The following commands are commands that can be recognized by the **MSD SUPER DISK DRIVE.**

INITIALIZE COMMAND: **"INITIALIZEØ"** or **"IØ"**

Purpose: To revert the disk drive back to its original condition when it was powered up.

The INITIALIZE command should be used when the disk drive has encountered an error that prevents it from performing any further operations. Because this command effectively "restarts" the computer, any operations or files that were open or in progress are terminated. Using a 1 instead of Ø will initialize drive 1 on the SD-2. The following formats will initialize the drive:

OPEN 15,8,15,"IØ" or
OPEN 15,8,15:PRINT#15,"IØ"

NEW COMMAND: **"NEWØ:diskette name,id"** or **"NØ:diskette name,id"**

Purpose: Erase the contents of the diskette and make it usable by the disk drive.

The diskette name is a user defined name that is placed on the disk for the convenience of user identification. The id is a two digit alphanumeric identifier that is placed in the directory as well as every block on the diskette. The id is used to make sure a diskette has not been changed in the course of writing to the disk (unless the id is the same on both disks). The id should be different for each diskette formatted or errors can be introduced onto a diskette if it is changed without initializing the disk drive. If 1 is substituted for Ø in the above command, the diskette in drive 1 will be formatted on the SD-2.

Any time a diskette is used for the first time (i.e. nothing has ever been stored on the diskette) it must be formatted so that the disk drive will recognize the disk. If a diskette has become unusable because of some diskette failure, it must be formatted again. The NEW command will erase the entire contents of the diskette and place the correct timing and block marks on the diskette. The BAM and directory are also created. The NEW command is used as follows:

```
OPEN 15,8,15,"NØ:TEST DISK,MS" or
OPEN 15,8,15,"NØ:PRACTICE,88"
```

If a diskette does not need to be reformatted, but it is desirable to erase the contents of the diskette, the same command can be used except the id is left off of the command:

```
OPEN 15,8,15,"NØ:TEST DISK" or
OPEN15,8,15:PRINT#15,"NØ:PRACTICE"
```

SCRATCH COMMAND: **"SCRATCHØ:file name"** or **"SØ:file name"**

Purpose: The SCRATCH command is used to erase a file or files from the diskette and thereby make additional space on the diskette.

The file name represents the directory file that is no longer desired on the diskette. Substituting a 1 for a Ø on the SD-2 will erase the specified file or files on drive 1. The following examples illustrate the SCRATCH command:

```
OPEN 15,8,15,"SØ:GOAWAY" or
OPEN 15,8,15:PRINT#15,"SØ:GONE"
```

It is possible to erase one file at a time or a number of files at one time. In order to erase more than one file at a time pattern matching and/or wild cards can be used. It is also possible to erase more than one file by using the syntax above and adding each file name to be erased after the program name with a comma seperating the file names. For example:

```
OPEN 15,8,15:PRINT#15,"SØ:TEXT,Ø:TEST,Ø:MUSIC"
```

would erase the three files TEXT, TEST and MUSIC from the diskette.

It is also possible to verify the number of files erased by reading the error channel after scratching the files (See Appendix B for an explanation of reading the error channel). The error channel will put the number of files erased in place of the track number.

COPY COMMAND: **"COPYØ:newfile=Ø:oldfile"** or **"CØ:newfile=Ø:oldfile"**

Purpose: To copy any program or file to another program or file on the diskette.

The COPY command is used to create a duplicate of an existing program or file on the diskette under a different name. In the command, newfile would represent the name of the file to be created while oldfile would be the file that is to be copied. Files can be copied onto the same diskette or in the case of the

SD-2 from one diskette to another. It is possible to create a file that is the combination of several files using the COPY command as given below:

"CØ:newfile=Ø:oldfile1,Ø:oldfile2,Ø:oldfile3"

The above command would combine oldfile1, oldfile2 and oldfile3 into one file on the diskette with the name newfile. Only data files can be successfully merged. If BASIC programs are combined, the BASIC within the computer cannot recognize anything except for the first program because of certain data stored at the end of a program file to indicate the end of a BASIC program. Examples of copying one program to another would be:

```
OPEN 15,8,15,"CØ:DUPLICATE=Ø:ORIGINAL"
OPEN 15,8,15:PRINT#15,"CØ:BACKUP=Ø:MASTER
OPEN 15,8,15:PRINT#15,"C1:DUPLICATE=Ø:ORIGINAL (SD-2 only)
```

An example of combining four files into one file using the COPY command is:

```
OPEN 15,8,15,"CØ:BIG=Ø:TEXAS,Ø:GIANT,Ø:OCEAN,Ø:ELEPHANT
```

RENAME COMMAND: **"RENAMEØ:newname=Ø:oldname"** or
"RØ:newname=Ø:oldname"

Purpose: To change the name of a program or file in the diskette directory.

Occasionally, it is desirable to change only the name of a program or file on the diskette. The RENAME command makes this possible. The newname is the name that you want the existing file on the diskette (oldname) to be changed to. Files can be renamed on drive 1 if so specified on the SD-2. It is not possible to RENAME any files that are currently opened.

Some RENAME command examples are as follows:

```
OPEN 15,8,15,"RØ:NEWFILE=Ø:OBSOLETE"
OPEN 15,8,15:PRINT#15,"RØ:REVISED=Ø:ORIGINAL
```

VALIDATE COMMAND: **"VALIDATEØ"** or
"VØ"

Purpose: To clean up the diskette after it has been used for a long period of time.

After a diskette has had files repeatedly saved and erased, the diskette accumulates a number of small gaps. The VALIDATE command will clean up the diskette to remove many of these gaps and provide more storage space. In addition, any data files which were

opened but were not properly closed will be erased from the directory providing additional storage area. Specifying a 1 instead of a Ø will validate drive 1 on the SD-2. The VALIDATE command is executed as follows:

```
OPEN 15,8,15,"VØ"
or
OPEN15,8,15:PRINT#15,"VØ"
```

CAUTION: If random files (these are special types of files the user can create and are explained in CHAPTER 3) have been created, they will also be removed from the diskette when this command is executed.

CLOSE COMMAND: CLOSE<file #>

Purpose: To close a channel of communications that has formerly been opened.

The file number is that file number that was used in the OPEN statement. Once a file that has been opened is no longer needed for data transfer, it must be properly closed. Closing the file causes the directory and BAM to be correctly updated. If the file is not properly closed, all data associated with that file will be lost.

It is very important that the data files be closed before the error channel is closed. If the error channel is closed before the data files, the other files will be closed by the disk operating system. A file is open as long as the LED on the disk drive is red. On the SD-2, an LED exists for each drive, and these LEDs indicate when a file on that given drive is open. If BASIC is being used, it will still think the files are open and will try writing to them. The opposite problem can also happen. If the BASIC program leads to an error condition, all files will be closed in BASIC but not on the disk drive. If this condition occurs, the following command should be entered:

```
CLOSE15:OPEN15,8,15:CLOSE15
```

in order to reinitialize the disk and retain the files that were opened.

SPECIAL DUAL DISK DRIVE COMMANDS

The SD-2 has a special duplicate command available for making fast copies of diskettes. Instead of performing a file copy like the COPY command, the DUPLICATE command copies each block from one diskette to the other. The DUPLICATE command also formats the destination disk before duplicating. The purpose of this command is to create backup programs, however, the DUPLICATE command will

not copy diskettes that are copy-protected. The DUPLICATE command will copy diskettes that have been made on Commodore models 2Ø31, 4Ø4Ø, 1540, or 1541. The format for the DUPLICATE command is:

DUPLICATE COMMAND **"DUPLICATE<dest drive>=<source drive>"**
or "D<dest drive>=<source drive>"

Purpose: To copy the entire contents of a source diskette to the destination diskette.

It is important to notice the order of the copy because the destination diskette is formatted and all previous data is lost. Placing write-protect tabs over the diskette to be copied will prevent inadvertantly destroying a good disk.

An example of the DUPLICATE command to copy from drive 1 to drive Ø is:

```
OPEN 15,8,15:PRINT#15,"DØ=1"
```

CHAPTER 3
FILE DATA HANDLING

As the user gets into more advanced applications, it becomes necessary to be able to store and save more than just programs. The **MSD SUPER DISK DRIVE** provides several methods for storing data on the disk. The next three sections explain the different types of files that can be used when using the **MSD SUPER DISK DRIVE.**

SECTION 3.1 - SEQUENTIAL FILES

Sequential files are those files that are stored and read as their name implies, sequentially from beginning to end. There are basically three different types of sequential files that can be used on the **MSD SUPER DISK DRIVE.** The first type of sequential file is the program file which is abbreviated in the directory of the diskette as PRG. The program file is the only sequential file type that can be loaded. BASIC uses sequential program files when storing or reading programs. The second and third types of sequential files are designated as sequential (SEQ) and user (USR) respectively. These two file types are for data handling. The sequential files must be opened just as the command channel was opened in the last chapter in order to read or write to the file. The syntax for opening a sequential file is:

OPEN <file #>,<device #>,<data channel>,"DR:file,type,direction"

The file number is a user assigned number between 1 and 255 or a predefined variable (usually the same number as the data channel for ease of remembering) that refers to the particular file that has been selected for any data transfers. The device number is usually 8 or a predefined variable. The data channel is a number between 2 and 14 that the user assigns as the channel that will be used for communicating. DR represents the drive number which is Ø for the SD-1 and either Ø or 1 for the SD-2. The file is the file name on the diskette to be opened. If the file does not exist it will be created in the case of a write or create an error condition in the case of a read (See Appendix B for an explanation of the error messages). The type represents the file type to be opened (PRG, SEQ, USR). The file type is abbreviated to its first character in the OPEN statement. The direction is either read or write which is designated as R or W respectively in the OPEN command. When a file has been opened, the LED relating to the particular drive will be red to indicate a file is open. The LED will remain red until all files on that drive are closed. Some examples of opening of sequential files are:

```
OPEN 5,8,5,"Ø:DATAFILE,S,R"

OPEN 2,8,2,"Ø:TEXT,P,W"
```

```
OPEN A,B,C,"Ø:"+A$+",U,R"
```

Just as the SAVE command has the capability of replacing an existing program, any existing program that has been opened for write can replace the existing program if the program name is preceded by the @ symbol. For instance:

```
OPEN 8,8,8,"@Ø:NEWONE,S,W"
```

will replace the existing file called NEWONE with the new data to be written.

Once a file has been opened for reading or writing, three different commands can be used to actually handle the data transfer. These three commands are the PRINT#, INPUT# and GET#. The PRINT# command is used for writing to the diskette while the INPUT# and GET# are used for reading from the diskette.

PRINT#file #,data

Purpose: The PRINT# command is used for writing data to the diskette.

The PRINT# (no spaces are allowed between the PRINT and the # sign) command, as explained in CHAPTER 2, is used to direct any output to the file that has been opened. The file number is that file that has been opened by the OPEN command. The data consists of variables and/or text that has been enclosed inside of quotation marks. The PRINT# command works like the PRINT command in BASIC. Any commas that are used to separate items on a line will result in spaces being stored to the diskette. Semicolons will keep spaces from getting stored to the diskette when used to separate items on a line. The absence of either a comma or semicolon will cause a carriage return (CR) to be stored on the diskette at the end of a PRINT# command.

Any data that is sent to the diskette using the PRINT# command will be stored sequentially byte by byte with all spaces, carriage returns and any other special character stored. As an example, consider the following program:

```
1Ø A$="THIS IS A"
2Ø B$="TEST"
3Ø OPEN 8,8,8,"Ø:TEST,S,W"
4Ø PRINT#8,A$,B$"OF THE DISK"
5Ø CLOSE8
6Ø END
```

The data will actually be stored in the following form:

```
1  2  3  4  5  6  7  8  9  1Ø  11  12  13  14  15  16  17  18  19
T  H  I  S     I  S     A
```

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
T	E	S	T	O	F		T	H	E		D	I	S	K	cr

The numbers represent the character position in the sequential file. The cr represents a carriage return which is inserted at the end of any PRINT# statement unless a semicolon or comma is used at the end. After the file is closed, a marker is placed at the end of the file.

The comma, carriage return and colon have special significance when stored to the disk. When used inside a string, they will be stored to the disk. When used as a separator between fields, the comma inserts spaces, while the carriage return and colon end the field and store a carriage return to the disk. For example, the statement:

```
PRINT#8,"NO, THANK YOU"
```

will put the comma inside the quotes onto disk. The statement:

```
PRINT#8,"NO","THANK YOU"
```

will cause spaces to be inserted between NO and THANK YOU instead of a comma. The importance of using the comma, carriage return, and colon on the diskette becomes apparent when using the INPUT# or GET# command to retrieve the data from the disk as explained below.

GET# COMMAND: GET#<file #>,<variable>

Purpose: To get data from the disk a byte at a time.

From time to time, it will be desirable to get data from the disk a byte (or character) at a time. The GET# command is used for that purpose. This command is useful because all data that is on the disk is read, including the comma, carriage return and the colon. The file # is the file number that has been opened while the variable is a single variable or a list of variables separated by commas. The variables can be string or numerical variables but string variables are recommended. If string data is encountered on the diskette when a number was requested, a BASIC error statement is generated. Since a string variable can be a number or character, it can always be read. Some examples of the GET# statement are:

```
GET#8, A$

GET#8, A$,B$,C$
```

The GET# statement is especially useful when the actual data content or structure on the disk is not known. Since every

character is received, disks that have had portions destroyed can be examined to possibly recover data that would not normally be recovered. For situations where a group of data is to be read and the data structure is known, the INPUT# statement is better suited. An example of the use of the GET# command is given below. This program will read the contents of the file that was stored in the program example for the PRINT# command above and display it on the screen.

```
10 OPEN 8,8,8,"TEST"
20 GET#8,A$:PRINT  A$;
30 IF ST=0 THEN 20
40 CLOSE 8
50 END
```

The ST in line 30 is the status byte that is in the computer. This byte will be 0 until the end of the file is reached.

INPUT# COMMAND: INPUT#<file #>,<variable>

Purpose: To retrieve data from the disk in groups rather than a character at a time.

The file # is the file number that was used in the OPEN command. The variable is a single variable or multiple variables separated by commas. The variables can be strings or numeric. In order to read a group of data from the diskette, it is necessary to be able to recognize the start and end of that group of data. This is accomplished through the use of separators as explained in the PRINT# section. The INPUT# command recognizes a comma, carriage return, and a colon as a separator between data fields. If a variable is to be read in from diskette, these separators will determine exactly how much data will come in each time.

As mentioned before, the data can be character strings or numbers. If the data is numeric, it is stored just as if the STR$ function had been performed on the numeric data. Numbers are stored with positive numbers containing a space as the first character while negative numbers have a minus sign as the first character. An example of the INPUT# instruction in conjunction with the PRINT# is provided below:

```
10 OPEN 8,8,8,"@0:DATAFILE,S,W"
20 FOR A=1 TO 10
30 PRINT#8,A
40 NEXT
50 CLOSE 8
60 OPEN 2,8,2,"DATAFILE"
70 INPUT#2,B:PRINT B
80 IF ST=0 THEN 70
90 CLOSE 2
100 END
```

The above example will write the numbers 1 through 10 to a sequential file called DATAFILE. Lines 70 and 80 will read the data from the disk and print it out.

SECTION 3.2 - RANDOM FILES

Sequential files are handy when working with continuous streams of data and program files, but quite often it is desirable to be able to access certain data without having to go sequentially through each byte. For this type of data, it is more desirable to use a random access method of data storage and retrieval. Random files provide the capability of obtaining data from within a file without reading the entire contents of the file. Another method of accessing data in a random fashion is to use relative files. Relative files will be explained in section 3.3. The decision to use random files versus relative files is determined by considering several differences between the two methods. Random files are generally more desirable when speed is a factor. Many machine language programs will utilize random files because of the speed improvement over relative files. The problem with random files is that the location of data using random files must be maintained by the program using the random files while relative file locations are maintained by the disk operating system. This is the primary reason for the speed difference. Random files can be more easily removed from the diskette inadvertently since the disk operating system does not maintain those files.

Random files are files that have been written to a certain designated location on the diskette. From the definitions given in Chapter 2, we learned that the disk is physically divided into 35 tracks with each track containing from 17 to 21 sectors or blocks. The following table indicates how many sectors are in each track:

TRACK NUMBER(S)	NUMBER OF SECTORS (BLOCKS)
1 to 17	21
18 to 24	19
25 to 30	18
31 to 35	17

Notice that adding the number of blocks in each sector results in a total number of 683 blocks which is the number of blocks available on a newly formatted diskette. If you consider the 19 blocks that make up the directory on track 18 and subtract that from the 683, the 664 bytes that remain corresponds to the amount of blocks available on a newly formatted disk as indicated by loading and listing the directory of a newly formatted disk! By using the random file commands it is possible to read from or write to any block on the diskette as well as determine which blocks are available for use. The following commands explain how to use the random file functions.

OPEN COMMAND: **OPEN<file#>,<device #>,<channel #>,"#"** or
OPEN<file#>,<device #>,<channel #>,"#<buffer #>

Purpose: To open a random file for reading or writing.

The file#, device #, and channel # have the same syntax as sequential files with only the '#' symbol used at the end of the command indicating that a random file is being opened. A minimum of two channels must be opened in order to use random files. The command channel is opened to channel 15 as was explained in earlier sections (i.e. OPEN 15,8,15). The second channel is opened according to the above syntax. The '#' symbol causes the disk to allocate a 256 byte buffer for the purpose of handling the desired block of data. If a buffer number is specified as in the second OPEN command above, the specified buffer will be allocated to that data channel. Ten buffers are available on the MSD SUPER DISK DRIVE (numbered Ø through 9). An example of each OPEN command is given below:

```
OPEN 8,8,8,"#"

OPEN 7,8,7,"#3"
```

NOTE: Do not specify "Ø:#" or "1:#"

BLOCK READ COMMAND:
"BLOCK-READ:"<channel #>;<drive#>;<track#>;<block#>

Purpose: To read a specific block of data from the disk.

The BLOCK-READ can be abbreviated to B-R. The file # and channel # are the file and channel that has been opened. The track # and block # indicate which 256 byte block is to be read. Executing this command causes the disk drive to move the specified block of data from the diskette to the buffer area specified in the OPEN command. The data can be read from the buffer area using the INPUT# and GET# commands. Only data that has been stored on that particular block will be read. Any unused bytes in the block will not be read. The sample program below indicates a method of using the BLOCK-READ command to read the contents of block 9 on track 5 and display the contents of that block on the screen:

```
10 OPEN 15,8,15
20 OPEN 8,8,8,"#"
30 PRINT#15,"B-R:"8;0;5;9
40 GET#8,A$
50 PRINT A$;
60 IF ST=0 THEN 40
70 PRINT"READ COMPLETE"
80 CLOSE8:CLOSE15
```

BLOCK WRITE COMMAND:
"BLOCK-WRITE:"<channel #>;<drive#>;<track#>;<block#>

Purpose: To write a block of data to a specified location on the disk.

The BLOCK-WRITE performs the opposite function of the BLOCK-READ command. The BLOCK-WRITE portion of the command can be abbreviated B-W. Execution of this command will cause the previously stored buffer information to be written to the specified location on the diskette. The information is transferred to the buffer through the channel other than the command channel that has been opened. This transfer is accomplished with the PRINT# command. The disk operating system keeps track of how many bytes are stored into the buffer and stores the byte count pointer into the first byte of the block on execution of the BLOCK-WRITE. This means that only 255 bytes can actually be written to or read from the block. The pointer takes up the first byte of the block. An example of a routine that will write data to the same block that is read in the BLOCK-READ example (track 5, block 9) is given below:

```
10 OPEN 15,8,15
20 OPEN 8,8,8,"#"
30 FOR A=1 TO 30
40 PRINT#8,"TESTING"
50 NEXT
60 PRINT#15,"B-W:"8;0;5;9
70 CLOSE8:CLOSE15
```

BLOCK ALLOCATE COMMAND:
"BLOCK-ALLOCATE:"<drive #>;<track#>;<block#>

Purpose: To determine if a particular block is free on the diskette and allocate it if it is free.

As mentioned earlier, the operating system does not maintain the diskette when BLOCK-READs and BLOCK-WRITEs are used. The user does have the ability to make sure a particular block is available by using the BLOCK-ALLOCATE command. The BLOCK-ALLOCATE command can be abbreviated B-A. This command makes it possible to use the BLOCK commands with a diskette that already has files on it. By checking the BAM, the command can determine if the specified block has been used. Since the BAM is updated each time a file is stored to diskette, files can be preserved. BLOCK-WRITEs do not update the BAM and therefore will not be recognized on the diskette unless a BLOCK-ALLOCATE command has been issued. All random files can be retained on the diskette until a VALIDATE command is executed. The VALIDATE command will not recognize any random file, and should therefore never be executed on a diskette that has such files.

If the BLOCK-ALLOCATE command determines that the specified block has already been used, an error message will be generated (See Appendix A for error message 65). The error message returns the next available track and block found on the diskette. The track and block returned does not get allocated unless the BLOCK-ALLOCATE command is issued for that track and block. The following command will allocate a block and write that block. If the block is already used it will write to the indicated available block.

```
10 OPEN 15,8,15:OPEN 8,8,8,"#"
20 PRINT#8,"THIS GOES INTO THE BUFFER"
30 T=5:S=9
40 PRINT#15,"B-A:"0;T;S
50 INPUT#15,A,A$,B,C
60 IF A=65 THEN T=B:S=C:GOTO 40
70 PRINT#15,"B-W:"8;0;T;S
80 PRINT"DATA WAS STORED IN TRACK:"T,"SECTOR:"S
90 CLOSE 8:CLOSE 15
100 END
```

Line 20 stores some data into the buffer while line 30 starts looking at track 5 block 9. Lines 40, 50, and 60 find a block that is available. Lines 70 and 80 actually write the buffer to the available block and print on the screen which track and block was finally found to store the buffer. If the program is repeated, a different block will be used each time the program is run since the previous occupied the earlier available block.

BLOCK-FREE COMMAND:
"BLOCK-FREE:"<drive #>;<track#>;<block#>

Purpose: To make a specified block on the diskette available for use.

The BLOCK-FREE command will perform similar to the SCRATCH command in that any specified block on the diskette can be made available. Instead of actually erasing the file, this command updates the BAM to indicate that the block is available for other files. The following command will free Track 5 block 9 for use:

```
OPEN8,8,8,"#":OPEN 15,8,15:PRINT#15,"B-F:"0,5,9:CLOSE8:CLOSE15
```

BUFFER-POINTER COMMAND
"BUFFER-POINTER:"channel#;location

Purpose: To allow random access of information within a block.

As mentioned in the BLOCK-WRITE command, a pointer is stored on each block of the diskette to indicate how many bytes were written

into the block. This buffer pointer also points to the next location that a piece of data is to be read from. The BUFFER-POINTER command makes it possible to move the pointer to any location within a given block making it possible to locate any byte inside a block. This command makes it easier to maximize the use of the space in the blocks. The BUFFER-POINTER in the command can be abbreviated B-P. As an example, the 32ND byte in the buffer can be pointed to and retrieved by the following program:

```
10 OPEN 15,8,15:OPEN 8,8,8,"#"
20 PRINT#15,"B-P:"8;32
30 GET#8,A$:PRINT A$
40 CLOSE 8:CLOSE 15
```

USER1 COMMAND:
"U1:"<channel#>;<drive#>;<track#>;<block#>

Purpose: To read a full 256 byte block from the diskette to the buffer.

The USER1 command is almost identical to the BLOCK-READ command except that the USER1 command will read the entire 256 byte block while the BLOCK-READ will only read the amount of data that has been stored into that block as indicated by the buffer-pointer. The USER1 command effectively forces the buffer-pointer to 255 so that the entire block is read.

The USER commands are specially designated commands that will be explained in more detail in the next chapter. The USER1 command is abbreviated to U1 or UA as both abbreviations are recognized by the disk operating system. The following example will get the entire 256 bytes from Track 5 Block 9 and display it to the screen:

```
10 OPEN 15,8,15: OPEN 8,8,8,"#"
20 PRINT#15,"U1:"8;0;5;9
30 GET#8,A$:PRINT A$;
40 IF ST=0 THEN 30
50 CLOSE 8:CLOSE 15
60 END
```

USER2 COMMAND
"U2:"<channel#>;<drive#>;<track#>;<block#>

Purpose: To write a block to the diskette without altering the buffer-pointer on the diskette.

The USER2 command (abbreviated U2 or UB) is very similar to the BLOCK-WRITE command in that it writes the contents of the buffer to the block on the disk. The difference is that the USER2 command does not change the buffer-pointer that is already on the diskette when the buffer is written to the diskette. This command is useful

when a programmer wishes to read a block of data into the buffer and modify some of the contents by using the BUFFER-POINTER command to find the data to be modified. The data can then be rewritten to the diskette with the USER2 command and the buffer-pointer on the diskette will be correct. If the BLOCK-WRITE command were used, it would be necessary to set the buffer-pointer to the number of bytes in the buffer before executing the BLOCK-WRITE. The following program illustrates the use of the USER2 command:

```
10 OPEN 15,8,15:OPEN 8,8,8,"#"
20 PRINT#15,"U1:"8;0;5;9
30 PRINT#15,"B-P:"8,32
40 PRINT#8,"A"
50 PRINT#15,"U2:"8;0;5;9
60 CLOSE 8:CLOSE 15
70 END
```

The above program reads Track 5 Block 9 into the buffer (line 20). Line 30 moves the pointer to the 32nd position while line 40 changes that byte to the character 'A'. Line 50 prints the buffer back to the disk. Even though the buffer pointer has been altered, the USER2 command makes sure the old buffer pointer is not altered on the diskette.

SECTION 3.3 - RELATIVE FILES

As mentioned in the last section, it is often desirable to be able to access different pieces of data on a direct basis rather than sequentially. The random files can accomplish this as long as the programmer maintains the files in his or her own program. Relative files are a little more flexible in that the disk operating system maintains the data on the disk. Relative files are slower than random files but the fact that the operating system maintains the files often makes up for the speed difference.

Relative files have two main components. These two components are the side sectors and the data blocks. The components are linked together to form a complete relative file. The data blocks consist of records that can be from 1 to 254 bytes in length. The total number of records can be contained in as many as 720 data blocks because the side sectors, of which there are 6, can each handle up to 120 data blocks. The result is that each diskette can contain up to 182,880 bytes of data in a file (120 pointers/side sector * 6 side sectors * 254 bytes per data block). The data block format consists of the first two bytes specifying the track and sector of the next data block. The next 254 bytes are actual data. Any empty record will have FF in the first byte with 00 in the rest of the record. The side sectors are used to reference all side sector locations as well as locations of the 120 data blocks relating to that side sector. The format is:

BYTE	DEFINITION
Ø,1	Track and sector of next side sector block.
2	Side sector number (Ø-5).
3	Record length.
4,5	Track and sector of first side sector (Ø).
6,7	Track and sector of second side sector (1).
8,9	Track and sector of third side sector (2).
1Ø,11	Track and sector of fourth side sector (3).
12,13	Track and sector of fifth side sector (4).
14,15	Track and sector of sixth side sector (5).
16-256	Track and sector pointers to 12Ø data blocks.

Relative files are created the first time they are opened. From then on that same file will be used until it is closed. A relative file can only be erased from a file using the SCRATCH command or reformatting the entire disk. The '@' sign which is often used with the SAVE command as a replace and save will NOT work with relative files. The syntax for opening a relative file is:

OPEN<file #>,<device#>,<channel#>,"<name>,L,"+CHR$(record length)

The name is the name that is to be assigned to the relative file and the record length specifies how many bytes are in each record. The record length and L are not required on files that already exist on the diskette. If an incorrect record size is specified for an existing file, an error message will be generated (See Appendix B for an explanation of error messages). The syntax for opening an existing relative file would be:

OPEN<file #>,<device#>,<channel#>,"name"

After a file has been opened, it is necessary to specify which record is to be accessed. The command for locating the file pointer to the correct record is:

PRINT<file #>,"P"CHR$(chan#)CHR$(rec#lo)CHR$(rec#hi)CHR$(position)

Because up to 72Ø records can exist in a file, it is necessary to specify a given record location in two bytes. The rec#lo and the rec#hi field identify the correct location of the record. The actual formula for determining what to put into these two bytes is RECORD# = record#hi * 256 + record#lo. The position represents the actual position in a record to start the data transfer.

An example of the creation of a relative file is given below:

```
1Ø OPEN 15,8,15
2Ø OPEN8,8,8,"Ø:TEST,L,"+CHR$(5Ø)
3Ø PRINT#15,"P"CHR$(8)CHR$(Ø)CHR$(4)CHR$(1)
4Ø PRINT#8,CHR$(255)
5Ø CLOSE8:CLOSE15
```

The above program will create a file called TEST that will contain

records that are 50 bytes in length. Line 30 will move the pointer to the first position in record number 1024 (the formula is RECORD#=256*4+0=1024). Notice that the pointer command is sent through channel 15 while data is sent, in this case, through channel 8. Since this is a newly created file that record will not actually exist. Because that record does not exist an error message (See Appendix B for error message 50) will be generated. This error message really serves as a warning to alert you that the record does not exist and you should not try to use GET# or INPUT# commands. Line 40 will not only cause an FF to be written into the 1024th record, but will also cause all records up to that point to get initialized. All necessary side sectors and data blocks will be created. If the disk does not have enough space for these data blocks, an error message 52 will be generated. The last line of the program will close the file.

If a relative file already exists, it is possible to open the file and either expand it or access it for data transfer. The file can be expanded but the record size must remain the same. To expand a file it is only necessary to specify a larger number of records in the PRINT statement like Line 30 above. An example of writing data to an existing relative file is given below:

```
10 OPEN 15,8,15
20 OPEN 2,8,6,"0:TEST"
30 GOSUB 1000
40 IF A=100 THEN STOP
50 PRINT#15,"P"CHR$(6)CHR$(100)CHR$(0)CHR$(1)
60 GOSUB 1000
70 IF A=50 THEN PRINT#2,1:GOTO50
80 IF A=100 THEN STOP
90 PRINT#2,"123456789"
100 PRINT#15,"P"CHR$(6)CHR$(100)CHR$(0)CHR$(20)
110 PRINT#2,"JOHN QWERTY"
120 CLOSE2:CLOSE15
130 END
1000 INPUT#15,A,A$,B$,C$
1010 IF (A=50) OR (A<20) THEN RETURN
1020 PRINT"FATAL ERROR: ";
1030 PRINT A,A$,B$,C$
1040 A=100:RETURN
```

Lines 10 and 20 open the command and a data channel respectively. Lines 30 and 40 check for possible errors. In this case the program makes sure the file already exists. The file pointer is moved in Line 50 to the 100th record position. Since no records exist at this time, an error 50 is generated. Lines 60, 70, and 80 check for the error and will create the 100 records. Line 90 prints the nine bytes of data to the first 9 locations in record 100. Line 100 moves the pointer to the 20th position in record 100 and line 110 prints the specified name into the record from that position.

It is important that data be written into the record sequentially from beginning to end because writing data at the beginning of the file destroys the rest of the record in the disk operating system memory. In the example above, data could not be written between positions 10 and 20 without destroying the name that has already been written into position 20 and beyond.

After the data has been written to any of the records, it is possible to read that data back at a later date. The example below illustrates a routine that will read a particular record from a relative file:

```
10 OPEN 15,8,15
20 OPEN 2,8,6,"0:TEST"
30 GOSUB 1000
40 IF A=100 THEN STOP
50 PRINT#15,"P"CHR$(6)CHR$(100)CHR$(0)CHR$(1)
60 GOSUB 1000
70 IF A=50 THEN PRINT A$
80 IF A=100 THEN STOP
90 INPUT#2,D$:PRINTD$,
100 PRINT#15,"P"CHR$(6)CHR$(100)CHR$(0)CHR$(20)
110 INPUT#2,E$:PRINTE$
120 CLOSE2:CLOSE15
130 END
1000 INPUT#15,A,A$,B$,C$
1010 IF (A=50) OR (A<20) THEN RETURN
1020 PRINT"FATAL ERROR: ";
1030 PRINT A,A$,B$,C$
1040 A=100:RETURN
```

This program will work in conjunction with the previous relative write routine to read the record that was previously stored. Lines 10 through 80 are about the same as the last example, except the file is expected to be on the disk or an error will result. Lines 90 - 110 actually read the record and display the contents to the screen. Notice that the carriage return that is sent to the disk after each PRINT# statement on the relative write routine is the separator for each field of the record.

If the file is to be written or read sequentially, it is not necessary to adjust the pointer to each record. The record pointer automatically starts at position 1 of the record if no other position has been defined. The pointer moves through the record as each field is read or written. As long as the pointer is not altered, it will point to the next field in the record.

CHAPTER 4
PROGRAMMING THE DISK CONTROLLER

The **MSD SUPER DISK DRIVE** is a "smart" peripheral meaning that it contains its own microprocessor and memory. It is possible for the advanced programmer to actually access the microprocessor and its memory to provide a wide number of applications. This chapter deals with the various commands as well as the architecture of the **MSD SUPER DISK DRIVE.** Before explaining the commands available for accessing the **SUPER DISK DRIVE,** it is necessary to explain the architecture of the disk drive itself.

The disk drive is operated by the 6511Q microprocessor, a 16K byte ROM based operating system, and 4K (6K for the SD-2) of random access memory. The 6511Q contains the necessary I/O lines for handling both serial and parallel IEEE communications as well as the disk control functions. The 6511Q also contains 192 bytes of internal RAM. The disk mechanism is a standard 5-1/4 inch disk drive with self-contained electronics for handling motor speed control and data manipulation. This approach allows a wide variety of commercially available disk drives to be used in the SUPERDRIVE. The memory organization is given in the table below:

LOCATION	FUNCTION
$0000-$003F	Job Cue Access
$0040-$00FF	6511Q Internal RAM
$0100-$013F	6511Q Control Lines
$0140-$3FFF	Not Used
$4000-$4FFF	SD-1 & SD-2 Buffer/OS Ram Area
$5000-$57FF	SD-2 Additional Buffer/OS Ram Area
$5800-$9FFF	Not Used
$A000-$BFFF	Hardware Control Area
$C000-$FFFF	ROM Operating System Area

The most useful area to the advanced programmer is the Buffer Ram area located between $4000 and $4FFF ($57FF on the SD-2). This area can actually be written into as well as the 6511Q internal RAM area with machine language level instructions and executed by the microprocessor in the disk. By down-loading special routines to execute within the disk drive and executing these routines, additional disk operations or program execution can be obtained. The method of handling data transfers to and from memory are referred to as memory commands. Three MEMORY commands exist for handling memory operations as well as some additional commands referred to as USER commands.

The following two sections describe these commands in detail. Some rules must be observed. The disk drive must be initialized using the INITIALIZE command the first time a sequence of MEMORY commands are sent to the drive. This should only be done once. Each of the three memory commands must be abbreviated and the use of a colon is not permitted in conjunction with these commands.

MEMORY-WRITE COMMAND:
"M-W"CHR$(adlo)CHR$(adhi)CHR$(# of bytes)CHR$(data)..

Purpose: Allows the disk memory to be directly written into.

The MEMORY-WRITE command allows up to 34 bytes to be transferred to the memory in the SUPERDRIVE. The adlo represents the decimal equivalent of the lower hexadecimal byte of the memory to be written to while adhi represents the decimal equivalent of the hexadecimal upper address byte. The number of bytes to transfer can be from one to 34 decimal and the data must be transferred in a decimal equivalent of the hexadecimal code. Each of the portions of the command are sent using the CHR$ instruction as illustrated below:

```
10 OPEN 15,8,15
20 PRINT#15,"M-W"CHR$(0)CHR$(64)CHR$(3)CHR$(165)CHR$(8)CHR$(96)
30 CLOSE 15
```

The above routine writes three bytes out to locations $4000 through $4002 (256*64+0=16384=$4000). The three bytes are 165 ($A5 which is a page zero LDA instruction), 8 (which is a loction of $0008), and 96 ($60 which is a return instruction). In effect, the above program when sent through this command and executed would cause the microprocessor in the disk drive to load its accumulator with the contents of location $0008 and then return control back to the disk drive.

The MEMORY-READ command is used to read an actual location in the memory map of the SUPERDRIVE. The command for reading the disk memory is:

MEMORY-READ COMMAND:
"M-R"CHR$(adlo)CHR$(adhi)

Purpose: To read a specific memory location within disk drive.

The only parameters that need to be specified are the lower byte of the address in decimal (adlo) and the upper address byte in decimal (adhi). The next byte that is read from channel #15 will be from the specified memory location. The following example illustrates how to read 11 consecutive bytes from location $FF00 to location $FF0A:

```
10 OPEN15,8,15,"I0"
20 FOR A=0 TO 10
30 PRINT#15,"M-R"CHR$(A)CHR$(255)
40 GET#15,A$:PRINT ASC(A$+CHR$(0));
50 NEXT
```

Once a program has been loaded into the disk drives memory, it is possible to execute that program by using the MEMORY-EXECUTE command. The format for the command is:

MEMORY-EXECUTE COMMAND:
"M-E"CHR$(adlo)CHR$(adhi)

Purpose: To execute a machine language program located in the memory of the disk drive.

Once a program has been downloaded to the disk memory, it can be executed using the MEMORY-EXECUTE command. The command specifies the upper address byte in decimal (adhi) and the lower address byte in decimal (adlo) at which the program begins. The use of this command requires that the end of the program that is to execute must be terminated with an RTS instruction so that control will be returned to the disk operating system.

The following command will execute a program that resides at location $5000:

```
OPEN15,8,15,"M-E"CHR$(0)CHR$(80)
```

Another alternative exists for executing programs using the BLOCK-EXECUTE command. If programs are stored on the diskette, they can be loaded and executed with this command. The format for the command is:

BLOCK-EXECUTE COMMAND:
"BLOCK-EXECUTE:"<channel#>;<drive#>;<track>;<block>

Purpose: To load a machine language program from a block on the diskette to the buffer memory and execute it.

The BLOCK-EXECUTE command, which can be abbreviated B-E, is similar to the other BLOCK commands discussed in the last section except that it will execute the block that has been transferred from the diskette to the buffer memory. The execution begins at first location of the buffer and ends when a machine language RTS instruction is encountered in the program. If a machine language program resides on Track 3 Sector 4, the following routine will load it into the disk buffer and execute it:

```
10 OPEN 15,8,15:OPEN 8,8,8,"#"
20 PRINT#15,"B-E:"8;0;3;4
30 CLOSE 8: CLOSE 15
```

Some other helpful commands exist for executing machine language programs in the disk drives memory. The commands are referred to as USER commands. A number of USER commands have already been discussed in their appropriate sections. The last section dealt with the USER1 and USER2 commands in conjunction with BLOCK-READs

and BLOCK-WRITEs respectively. The UI+ and UI- commands are used in conjunction with changing the speed of the serial communications bus between the VIC-20 and the Commodore 64. The VIC-20 is actually capable of loading data from the disk at a faster rate than the Commodore 64. The MSD SUPER DISK DRIVE powers on to the speed of the Commodore 64. If a VIC-20 is being used with the MSD SUPER DISK DRIVE, the speed can be changed using the UI- command. The other USER command previously discussed was the U; or UJ command which is a jump to the power-up vector that serves as a software reset to the disk drive. In addition to these commands, there exists seven other USER commands that actually cause a program to jump to certain places in the buffer memory for execution of different programs that may have been loaded into the buffer. The commands are used by opening the error channel and sending the command using a PRINT# command. An example of the use of a USER command is:

```
OPEN 15,8,15
PRINT#15,"U8"
```

A list of the different USER commands as well as a brief explanation of the command is presented below:

U1 or UA	BLOCK-READ without changing the buffer pointer
U2 or UB	BLOCK-WRITE without changing the buffer pointer
U3 or UC	Jump to $4200
U4 or UD	Jump to $4203
U5 or UE	Jump to $4206
U6 or UF	Jump to $4209
U7 or UG	Jump to $420C
U8 or UH	Jump to $420F
U9 or UI	Jump to ($FFFA) This is the NMI vector.
U: or U; or UJ	Jump to power-up vector
UI+	Set Commodore 64 serial speed
UI-	Set VIC-20 serial speed

CHAPTER 5
CHANGING THE DISK DRIVE DEVICE NUMBER

The device number on the **MSD SUPER DISK DRIVE** comes from the factory selected as device number 8, drive number Ø (and 1 in the case of the SD-2). This is the usual device number, but if more than one disk drive is to be used with the system, it is necessary to give each disk drive a different device number or drive number. With the disk drive unit it is not possible to change the drive number but the device number can be changed using either software or hardware. The disk drive determines its device number from a jumper that is located on the printed circuit board inside the disk drive. At the time the disk drive is powered on, the jumper is "read" by the microprocessor and the number is stored in a specific memory location. This makes it possible to change the device number by changing the jumper or by using a MEMORY-WRITE command to change the memory location that has the device number stored in it. Changing the device number by changing the jumper will cause the disk drive to always power on with that number. Changing the device number with the MEMORY-WRITE command in a program will mean that the number must be changed each time the disk drive is turned on. Each method has its own applications and is detailed below.

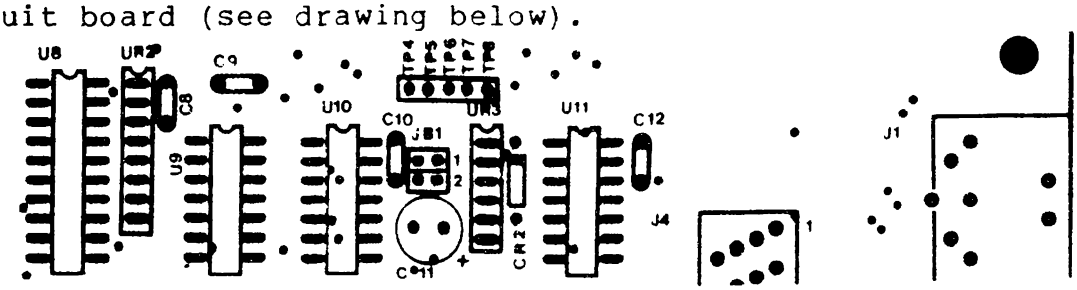
CHANGING THE DEVICE NUMBER BY THE JUMPER (HARDWARE METHOD)

In order to change the device number by the hardware method several steps must be followed:

1. Turn off the disk drive and remove all cables from the drive.

2. Remove the screws from the disk drive cover and remove the cover.

3. Locate the jumper block JB1 at the top rear of the printed circuit board (see drawing below).



4. A jumper is located on JB1-1 and a jumper is located on JB1-2 when it is shipped from the factory. This is the configuration for the disk to respond as device 8. If JB1-1 is unplugged and JB1-2 is installed, the device number will become 9. If JB1-2 is removed but JB1-1 is installed, the device number will be 1Ø. Removing both jumpers will set the device number to 11.

5. Reinstall the cover and fasten it securely with the screws that were removed earlier.

6. Reconnect the cables and turn the disk drive power switch back on. The drive is now ready to be used.

CHANGING THE DEVICE NUMBER BY SOFTWARE

The device number is changed by performing a MEMORY-WRITE to locations $ØØ77 and $ØØ78. The MEMORY-WRITE command for changing the device number is performed after the error channel has been opened and has the following format:

PRINT#<file#>,"M-W"CHR$(119)CHR$(Ø)CHR$(2)CHR$(dv+32)CHR$(dv+64)

The dv represents the device number that is desired. An example of a simple routine for changing the device number to device number 9 is given below:

```
1Ø OPEN 15,8,15
2Ø PRINT#15,"M-W"CHR$(119)CHR$(Ø)CHR$(2)CHR$(9+32)CHR$(9+64)
3Ø END
```

It is usually desirable to change the device number in hardware unless a temporary change is all that is desired. In order to use the software method, only one drive can be powered on, its device number changed, then another drive powered on and its device number changed until all drives are on. If this procedure is not followed, the disk drives will conflict with each other.

APPENDIX A
DISK COMMAND SUMMARY (EXCEPT BASIC 4.Ø COMMANDS)

The following list is a summary of each command that is available on the MSD SUPER DISK DRIVE (See APPENDIX C for BASIC 4.Ø commands). The < and > symbols are used to indicate that the programmer must supply the information between the symbols. **The < and > symbols are not actually a part of the command.** A device number of 8 is assumed for the disk drive.

COMMAND	COMMAND FORMAT
BLOCK-ALLOCATE	"B-A:"Ø;<track>;<block>
BLOCK-EXECUTE	"B-E:"<channel>;Ø;<track>;<block>
BLOCK-FREE	"B-F:"Ø;<track>;<block>
BLOCK-READ	"B-R:"Ø;<channel>;<track>;<block>
BLOCK-WRITE	"B-W:"Ø;<channel>;<track>;<block>
BUFFER-POINTER	"B-P:"<channel>;<location>
CLOSE	CLOSE<file#>
COPY	"CØ:<newfile>=Ø:<oldfile>"
DUPLICATE	"DØ=1" (SD-2 command only)
GET#	GET#<file#>,<variable>
INITIALIZE	"IØ"
INPUT#	INPUT#<file#>,<variable>
LOAD	LOAD "Ø:<file name>",8
MEMORY-EXECUTE	"M-E"CHR$(adlo)CHR$(adhi)
MEMORY-READ	"M-R"CHR$(adlo)CHR$(adhi)
MEMORY-WRITE	"M-W"CHR$(adlo)CHR$(adhi)CHR$(#bytes)CHR$(data)
NEW	"NØ:<diskette name>,<id>"
OPEN	OPEN<file#>,8,<sec adr>,"<text string>"
POSITION	"P"CHR$(channel)CHR$(reclo)CHR$(rechi)
PRINT#	PRINT#<file#>,"<text string>"
RENAME	"RØ:<newname>=Ø:<oldname>"
SAVE	SAVE "Ø:<file name>",8
SCRATCH	"SØ:<program name>"
USER1	"U1:"<channel>;Ø;<track>;<block>
USER2	"U2:"<channel>;Ø;<track>;<block>
USER COMMANDS	"U<number>:"
VERIFY	VERIFY "Ø:<program name>",8
VALIDATE	"V"

APPENDIX B
ERROR MESSAGES

Whenever an error occurs on the **MSD SUPER DISK DRIVE,** the LED on the front of the disk drive will start flashing. The disk drive will not send the error message to the computer unless it has been requested. In order to request an error message to determine what the error is, it is necessary to OPEN the error channel and read the error message. This can be accomplished by the following simple routine:

```
10 OPEN 15,8,15
20 INPUT#15,A,A$,B$,C$
30 PRINT A,A$,B$,C$
40 CLOSE15
50 END
```

If BASIC 4.0 is being used, it is possible to print the variable DS$ to get a display of contents of the error channel.

A list of each error message as well as an explanation of the error message is provided below. The format of the error meesage is as follows:

00,OK,00,00

Where the first field is the error number, the second field is the description of the error, and the last two fields represent the track and sector respectively where it is applicable.

0 OK
This error message is really not an indication of an error and will occur only if the error channel is read when the LED is not flashing.

1 FILES SCRATCHED
This is also a non-error condition. Reading the error channel after a file or files have been SCRATCHed will return with this message as well as the number of files that have been scratched in the track field.

2-19 Unused error message channels

20 READ ERROR (BLOCK HEADER NOT FOUND)
The disk controller is not able to locate the header of the block that has been requested. This can be caused by a bad header on the diskette or an illegal sector number being specified.

21 READ ERROR (NO SYNC CHARACTER)
The disk controller was not able to detect a sync mark on the desired track of the diskette. A number of things can cause this problem such as no diskette in the drive, improperly seated diskette, unformatted diskette, or some disk drive malfunction.

22 READ ERROR (DATA BLOCK NOT PRESENT)
This error results when the disk drive has been requested to read or verify a data block that was not properly written. The error results from an illegal track or sector number being specified in one of the BLOCK commands.

23 READ ERROR (CHECKSUM ERROR IN DATA BLOCK)
This error results from an error in one or more of the data bytes in the block. The checksum that was stored with the data on the diskette does not match the checksum generated by the data that was read from the diskette.

24 READ ERROR (BYTE DECODE ERROR)
Diskette did not format correctly. Problem can be caused by defective diskette media or disk drive motor speed.

25 WRITE ERROR (WRITE-VERIFY ERROR)
This error is the result of a mismatch between the data that was written on the diskette and the data that was in memory to be written to the diskette. The error occurs after a write.

26 WRITE PROTECT ON
This error message is generated any time the disk drive is requested to write data to the diskette and the write protect switch has been depressed. Each diskette has a write protect notch that must not be covered or this error will result.

27 READ ERROR (CHECKSUM ERROR IN HEADER)
The header has been found on the diskette but the checksum generated while reading the header is incorrect.

29 DISK ID MISMATCH
Whenever a diskette is inserted into the drive and initialized the diskette id is recorded by the controller. If another disk is inserted and not initialized this error will occur. A bad header can also produce this error.

30 SYNTAX ERROR (GENERAL SYNTAX)
This error results from the transfer of a command that the disk drive can not interpret completely. The command should be examined for possible errors and corrected before transferring again.

31 SYNTAX ERROR (INVALID COMMAND)
This is a result of the disk drive not recognizing the command as a legitimate command.

32 SYNTAX ERROR (LONG LINE)
The command sent to the disk drive is more than 40 characters.

33 SYNTAX ERROR (INVALID FILE NAME)
This error will result when pattern matching has been used incorrectly.

34 SYNTAX ERROR (NO FILE GIVEN)
The file name has not been given or an error exists in the portion of the file name specified.

35-38 Unused error message channels

39 SYNTAX ERROR (INVALID COMMAND)
This error can result if a command sent to the command channel is not recognized by the disk drive.

5Ø RECORD NOT PRESENT
This error message occurs in conjunction with relative files or the disk reading past the last record through the INPUT# or GET# commands. This error serves to warn that a GET# or INPUT# should not be attempted after this error message. The error message can be used in conjunction with a PRINT# statement to expand a relative file.

51 OVERFLOW IN RECORD
If the PRINT# statement exceeds the record boundary this error results. Information is truncated. This error often occurs when the carriage return is not considered in the count of the total number of characters in the record.

52 FILE TOO LARGE
If a disk overflow will occur as a result of the record position in a relative file this error will result.

53-59 Not used

6Ø WRITE FILE OPEN
This error results from trying to OPEN a file for reading that has not been closed after writing.

61 FILE NOT OPEN
If a file is being accessed that has not been OPENed, this error will result.

62 FILE NOT FOUND
This error results when the requested file is not in the directory of the diskette.

63 FILE EXISTS
The file name that is being created already exists on the diskette.

64 FILE TYPE MISMATCH
The file type specified does not match the file type found in the directory of the diskette.

65 NO BLOCK
If the BLOCK-ALLOCATE command is used on a block that has already been allocated, this error results. The parameters returned indicate the track and sector number of the next available track

and sector. A number of zero for the track and sector indicates that no other higher track or sector is available.

66 ILLEGAL TRACK AND SECTOR
The disk has tried to access a track or sector that does not exist.

67 ILLEGAL SYSTEM T OR S
This error message indicates an illegal system track or sector.

68-69 Not used

70 NO CHANNEL (AVAILABLE)
The requested channel is not available or all channels are currently being used.

71 DIRECTORY ERROR
This error results when the BAM does not match the internal count. Something has happened to the BAM image in memory or the BAM allocation is at fault. By sending an INITIALIZE command to the disk drive this error can be corrected. If some files were active they may be terminated by this initialization.

72 DISK FULL
The disk is full or the maximum number of entries are in the directory.

73 DOS MISMATCH
In the case where a particular disk is only read compatible, this error will result when an attempt is made to write to the diskette.

74 DRIVE NOT READY
This error results when an attempt is made to access the disk drive without any diskette in the drive.

APPENDIX C
BASIC 4.Ø COMMANDS

A set of commands exist for use in conjunction with BASIC 4.Ø. On computers that have the BASIC 4.Ø commands, the MSD superdrive will respond to those commands as well as the BASIC 3.Ø commands. These commands are identified in this section along with the appropriate BASIC 3.Ø command. For a more detailed explanation of the command, refer to the section that explains the comparable BASIC 3.Ø command. The < and > symbols are used to indicate that the programmer must supply the information between those symbols. Those symbols are not a part of the command.

BASIC 3.Ø COMMAND	BASIC 4.Ø COMMAND
LOAD"Ø:<program name>",8	DLOAD"<program name>",DØ
LOAD"$Ø",8:LIST	DIRECTORY or DI<shifted R>
LOAD"*",8:RUN	shifted RUN/STOP
SAVE"Ø:<program name>",8	DSAVE"<program name>",DØ

The commands that are sent to the error channels are different in many cases. One big difference is the fact that an error message number can be read without opening the error channel. The command for reading the error message number in BASIC 4.Ø is ?DS$ or ?DS. The other commands that are sent through the command channel are (assume that file #15 is opened):

BASIC 3.Ø COMMAND	BASIC 4.Ø COMMAND
PRINT#15,"IØ"	PRINT#15,"IØ"
PRINT#15,"NØ:<disk name>,<id>"	HEADER"<disk name>",DØ,I<id>,U8
PRINT#15,"SØ:<file name>"	SCRATCH"<file name>",U8
PRINT#15,"CØ:<nfile>=Ø:<ofile>"	COPY DØ,"<ofile>" TO "<nfile>",U8
PRINT#15,"CØ:<nfile>=<of1>,<of2>"	CONCAT DØ,"<of1>" TO "<nfile>",U8
PRINT#15,"DØ=1"	BACKUP D1 TO DØ
PRINT#15,"RØ:<new>=<old>"	RENAME DØ,"<old>" TO "<new>",U8
PRINT#15,"VØ"	COLLECT DØ,U8

In order to open and close a channel as well as specify the position pointer in a relative file using BASIC 4.Ø the following

commands apply:

DOPEN<file#>,"<file name>,DØ,L<record length>,8,<mode>

DCLOSE<file#>

RECORD<file#>,<record#>,<byte position>

****

SD-1 WARNING

****

This equipment generates, and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

Reorient the receiving antenna
Relocate the computer with respect to the receiver
Move the computer away from the receiver
Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:

"How to Identiify and Resolve Radio-Television Interference Problems".

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

The MSD Super Disk Drive cabinet will be warm when in operation. This is normal and will not effect operation. A special heat dissipater has been installed which transfers the heat to the entire cabinet thus keeping the electronic circuitry cool. Please do not block air flow from the vents located on the side and bottom of the cabinet.

****

SD-2 WARNING

****

This equipment generates, uses and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

The MSD Super Disk Drive cabinet will be warm when in operation. This is normal and will not effect operation. A special heat dissipater has been installed which transfers the heat to the entire cabinet thus keeping the electronic circuitry cool. Please do not block air flow from the vents located on the side and bottom of the cabinet.

MSD SYSTEMS, INC.
AUTHORIZED SERVICE CENTERS
September 18, 1984

The following locations are authorized by MSD Systems Inc. to perform non warranty repairs on the MSD Systems, Inc. SD-1 and SD-2 Super Disk Drives.

AA Computer Exchange
2726 Park St.
Jacksonville, FL 32205
(904) 384-6520

CAYCO Industries, Inc.
516 S.E. Coconut Isle
Fort Lauderdale, FL 33301
(305) 525-5100

Computer Outlet
5857 Mission Gorge Rd.
San Diego, CA 92120
(619) 282-6200

Computovision
16725 Bellflower Blvd.
Bellflower, CA 90706
(213) 920-7811

Digital Enterprises, Inc.
11331 E. 23rd St.
Independence, MI 64052
(816) 254-9152

O.K. T.V. and Appliance
931 University Ave
Honolulu, HI 96826
(808) 941-9666

Petry Products
409 South Leroy
Fenton, MI 48430
(313) 629-9090

H.B.H.
224 W. Main St.
Collinsville, IL 62234
(618) 344-7912

Yoder Service
6512 Truman Lane
Falls Church, VA 22043
(703) 241-0142

Skyles Electric Works
231 E. South Whisman
Mountain View, CA 94041
(415) 965-1735

Lakes Consumer Electronic
3232 S. Main St.
Akron, OH 44319
(216) 644-3194

Neighborhood T.V. Service
82 Carleton Ave
E. Islip, NY 11730
(516) 277-3591

Bytes & Bits
17 Diamond Lane
Columbia, SC 29210
(803) 772-9044

Ramtek
14034 Lambert Rd.
Whittier, CA 90605
(213) 941-1791

The Commodore Connection
1013 N. Johnson St.
Bay City, MI 48706
(517) 893-6999

Brought to you by:

https://www.facebook.com/groups/commodoreinternationalhistoricalsociety

commodore international historical society